

# On Kernelizing Mahalanobis Distance Learning Algorithms

Ratthachat Chatpatanasiri

RATTHACHAT.C@STUDENT.CHULA.AC.TH

Teesid Korsrilabutr

G48TKR@CP.ENG.CHULA.AC.TH

Pasakorn Tangchanachaianan

PASAKORN.T@STUDENT.CHULA.AC.TH

Boonserm Kijsirkul

BOONSERM.K@CHULA.AC.TH

Department of Computer Engineering, Chulalongkorn University, Pathumwan, Bangkok, Thailand.

## Abstract

This paper focuses on the problem of kernelizing an existing supervised Mahalanobis distance learner. The following features are included in the paper. Firstly, three popular learners, namely, “neighborhood component analysis”, “large margin nearest neighbors” and “discriminant neighborhood embedding”, which do not have kernel versions are kernelized in order to improve their classification performances. Secondly, an alternative kernelization framework called “KPCA trick” is presented. Implementing a learner in the new framework gains several advantages over the standard framework, e.g. no mathematical formulas and no reprogramming are required for a kernel implementation, the framework avoids troublesome problems such as singularity, etc. Thirdly, while the truths of representer theorems are just assumptions in previous papers related to ours, here, representer theorems are formally proven. The proofs validate both the kernel trick and the KPCA trick in the context of Mahalanobis distance learning. Fourthly, unlike previous works which always apply brute force methods to select a kernel, we investigate two approaches which can be efficiently adopted to construct an appropriate kernel for a given dataset. Finally, numerical results on various real-world datasets are presented.

## 1. Introduction

Recently, many Mahalanobis distance learners are invented (Chen et al., 2005; Goldberger et al., 2005; Weinberger et al., 2006; Yang et al., 2006; Sugiyama, 2006; Yan et al., 2007; Zhang et al., 2007; Torresani & Lee, 2007; Xing et al., 2003). These recently proposed learners are carefully designed so that they can handle a class of problems where data of one class form multi-modality where classical learners such as principal component analysis (PCA) and Fisher discriminant analysis (FDA) cannot handle. Therefore, promisingly, the new learners usually outperform the classical learners on experiments reported in recent papers. Nevertheless, since learning a Mahalanobis distance is equivalent to learning a linear map, the inability to learn a non-linear transformation is one important limitation of all Mahalanobis distance learners.

As the research in Mahalanobis distance learning has just recently begun, several issues are left open such as (1) some efficient learners do not have non-linear extensions, (2) the *kernel trick* (Schölkopf & Smola, 2001), a standard non-linearization method, is not fully automatic in the sense that new mathematical formulas have to be derived and new programming codes have to be implemented; this is not convenient to non-experts, (3) existing algorithms “assume” the truth of the *representer theorem* (Schölkopf & Smola, 2001, Chapter 4); however, to our knowledge, there is no formal proof of the theorem in the context of Mahalanobis distance learning, and (4) the problem of how to select an efficient

kernel function has been left untouched in previous works; currently, the best kernel is achieved via a brute-force method such as cross validation.

In this paper, we highlight the following key contributions:

- Three popular learners recently proposed in the literatures, namely, *neighborhood component analysis* (NCA) (Goldberger et al., 2005), *large margin nearest neighbors* (LMNN) (Weinberger et al., 2006) and *discriminant neighborhood embedding* (DNE) (Zhang et al., 2007) are kernelized in order to improve their classification performances with respect to the kNN algorithm.
- A *KPCA trick* framework is presented as an alternative choice to the kernel-trick framework. In contrast to the kernel trick, the KPCA trick does not require users to derive new mathematical formulas. Also, whenever an implementation of an original learner is available, users are not required to re-implement the kernel version of the original learner. Moreover, the new framework avoids problems such as singularity in eigen-decomposition and provides a convenient way to speed up a learner.
- Two representer theorems in the context of Mahalanobis distance learning are proven. Our theorems justify both the kernel-trick and the KPCA-trick frameworks. Moreover, the theorems validate kernelized algorithms learning a Mahalanobis distance in any separable Hilbert space and also cover kernelized algorithms performing dimensionality reduction.
- The problem of efficient kernel selection is dealt with. Firstly, we investigate the *kernel alignment* method proposed in previous works (Lanckriet et al., 2004; Zhu et al., 2005) to see whether it is appropriate for a kernelized Mahalanobis distance learner or not. Secondly, we investigate a simple method which constructs an unweighted combination of base kernels. A theoretical result is provided to support this simple approach. Kernel constructions based on our two approaches require much shorter running time when comparing to the standard cross validation approach.
- As kNN is already a non-linear classifier, there are some doubts about the usefulness of kernelizing Mahalanobis distance learners (Weinberger et al., 2006, pp. 8). We provide an explanation and conduct extensive experiments on real-world datasets to prove the usefulness of the kernelization.

## 2. Background

Let  $\{\mathbf{x}_i, y_i\}_{i=1}^n$  denote a training set of  $n$  labeled examples with inputs  $\mathbf{x}_i \in \mathbb{R}^D$  and corresponding class labels  $y_i \in \{c_1, \dots, c_p\}$ . Any Mahalanobis distance can be represented by a symmetric positive semi-definite (PSD) matrix  $M \in \mathbb{S}_+^D$ . Here, we denote  $\mathbb{S}_+^D$  as a space of  $D \times D$  PSD matrices. Given two points  $\mathbf{x}_i$  and  $\mathbf{x}_j$ , and a PSD matrix  $M$ , the Mahalanobis distance with respect to  $M$  between the two points is defined as  $\|\mathbf{x}_i - \mathbf{x}_j\|_M = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T M (\mathbf{x}_i - \mathbf{x}_j)}$ . Our goal is to find a PSD matrix  $M^*$  that minimizes a reasonable objective function  $f(\cdot)$ :

$$M^* = \arg \min_{M \in \mathbb{S}_+^D} f(M). \quad (1)$$

Since the PSD matrix  $M$  can be decomposed to  $A^T A$ , we can equivalently restate our problem as learning the best matrix  $A$ :

$$A^* = \arg \min_{A \in \mathbb{R}^{d \times D}} f(A). \quad (2)$$

Note that  $d = D$  in the standard setting, but for the purpose of dimensionality reduction we can learn a low-rank projection by restricting  $d < D$ . After learning the best linear map  $A^*$ , it will be used by kNN to compute the distance between two points in the transformed space as  $(\mathbf{x}_i - \mathbf{x}_j)^T M^* (\mathbf{x}_i - \mathbf{x}_j) = \|A^* \mathbf{x}_i - A^* \mathbf{x}_j\|^2$ .

In the following subsections, three popular algorithms, whose objective functions are mainly designed for a further use of kNN classification, are presented. Despite their efficiency and popularity, the three algorithms do not have their kernel versions, and thus in this paper we are primarily interested in kernelizing these three algorithms in order to improve their classification performances.

## 2.1 Neighborhood Component Analysis (NCA) Algorithm

The original goal of NCA (Goldberger et al., 2005) is to optimize the *leave-one-out* (LOO) performance on training data. However, as the actual LOO classification error of kNN is a non-smooth function of the matrix  $A$ , Goldberger et al. propose to minimize a stochastic variant of the LOO kNN score which is defined as follows:

$$f^{NCA}(A) = - \sum_i \sum_{y_j=c_i} p_{ij}, \quad (3)$$

where

$$p_{ij} = \frac{\exp(-\|A\mathbf{x}_i - A\mathbf{x}_j\|^2)}{\sum_{k \neq i} \exp(-\|A\mathbf{x}_i - A\mathbf{x}_k\|^2)}, \quad p_{ii} = 0.$$

Optimizing  $f^{NCA}(\cdot)$  can be done by applying a gradient based method. One major disadvantage of NCA, however, is that  $f^{NCA}(\cdot)$  is not convex, and the gradient based methods are thus prone to local optima.

## 2.2 Large Margin Nearest Neighbor (LMNN) Algorithm

In LMNN (Weinberger et al., 2006), the output Mahalanobis distance is optimized with the goal that *for each point, its  $k$ -nearest neighbors always belong to the same class while examples from different classes are separated by a large margin*.

For each point  $\mathbf{x}_i$ , we define its  $k$  *target neighbors* as the  $k$  other inputs with the same label  $y_i$  that are closest to  $\mathbf{x}_i$  (with respect to the Euclidean distance in the input space). We use  $w_{ij} \in \{0, 1\}$  to indicate whether an input  $\mathbf{x}_j$  is a target neighbor of an input  $\mathbf{x}_i$ . For convenience, we define  $y_{ij} \in \{0, 1\}$  to indicate whether or not the labels  $y_i$  and  $y_j$  match. The objective function of LMNN is as follows:

$$f^{LMNN}(M) = \sum_{i,j} w_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|_M^2 + c \sum_{i,j,l} w_{ij}(1 - y_{il}) \left[ 1 + \|\mathbf{x}_i - \mathbf{x}_j\|_M^2 - \|\mathbf{x}_i - \mathbf{x}_l\|_M^2 \right]_+,$$

where  $[\cdot]_+$  denotes the standard hinge loss:  $[z]_+ = \max(z, 0)$ . The term  $c > 0$  is a positive constant typically set by cross validation. The objective function above is convex<sup>1</sup> and has

---

1. There is a variation on LMNN called “large margin component analysis” (LMCA) (Torresani & Lee, 2007) which proposes to optimize  $A$  instead of  $M$ ; however, LMCA does not preserve some desirable properties, such as convexity, of LMNN, and therefore the algorithm “Kernel LMCA” presented there is different from “Kernel LMNN” presented in this paper.

two competing terms. The first term penalizes large distances between each input and its target neighbors, while the second term penalizes small distances between each input and all other inputs that do not share the same label.

### 2.3 Discriminant Neighborhood Embedding (DNE) Algorithm

The main idea of DNE (Zhang et al., 2007) is quite similar to LMNN. DNE seeks a linear transformation such that neighborhood points in the same class are squeezed but those in different classes are separated as much as possible. However, DNE does not care about the notion of margin; in the case of LMNN, we want every point to stay far from points of other classes, but for DNE, we want the average distance between two neighborhood points of different classes to be large. Another difference is that LMNN can learn a full Mahalanobis distance, i.e. a general *weighted* linear projection, while DNE can learn only an *unweighted* linear projection.

Similar to LMNN, we define *two* sets of  $k$  target neighbors for each point  $\mathbf{x}_i$  based on the Euclidean distance in the input space. For each  $\mathbf{x}_i$ , let  $Neig^I(i)$  be the set of  $k$  nearest neighbors having the same label  $y_i$ , and let  $Neig^E(i)$  be the set of  $k$  nearest neighbors having different labels from  $y_i$ . We define  $w_{ij}$  as follows

$$w_{ij} = \begin{cases} +1, & \text{if } j \in Neig^I(i) \vee i \in Neig^I(j), \\ -1, & \text{if } j \in Neig^E(i) \vee i \in Neig^E(j), \\ 0, & \text{otherwise.} \end{cases}$$

The objective function of DNE is:

$$f^{DNE}(A) = \sum_{i,j} w_{i,j} \|A\mathbf{x}_i - A\mathbf{x}_j\|^2.$$

which can be reformulated (up to a constant factor) to be

$$f^{DNE}(A) = \text{trace}(AX(D - W)X^T A^T),$$

where  $W$  is a symmetric matrix with elements  $w_{ij}$ ,  $D$  is a diagonal matrix with  $D_{ii} = \sum_j w_{ij}$  and  $X$  is the matrix of input points  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ . It is a well-known result from spectral graph theory (von Luxburg, 2007) that  $D - W$  is symmetric but is not necessarily PSD. To solve the problem by eigen-decomposition, the constraint  $AA^T = I$  is added (recall that  $A \in \mathbb{R}^{d \times D}$  ( $d \leq D$ )) so that we have the following optimization problem:

$$A^* = \arg \min_{AA^T=I} \text{trace}(AX(D - W)X^T A^T). \quad (4)$$

## 3. Kernelization

In this section, we focus on two kernelization frameworks going to non-linearize the three algorithms presented in the previous section. First, the standard kernel trick framework is presented. Next, the *KPCA trick* framework which is an alternative to the kernel-trick framework is presented. Kernelization in this new framework is conveniently done with an application of *kernel principal component analysis* (KPCA). Finally, representer

theorems are proven to validate all applications of the two kernelization frameworks in the context of Mahalanobis distance learning. Note that in some previous works (Chen et al., 2005; Globerson & Roweis, 2006; Yan et al., 2007; Torresani & Lee, 2007), the validity of applications of the the kernel trick has not been proven.

### 3.1 Historical Background

After finishing writing the current paper, we just knew that this name was first appeared in the paper of Chapelle and Schölkopf (2001) who first applied this method to invariant support vector machines; moreover, it is appeared to us that the KPCA trick has been known to some researchers (private communication to some ECML reviewers). Without knowing about this fact, we reinvented the framework and, coincidentally, called it “KPCA trick” ourselves. Nevertheless, we will shown in Section 5.1 that, in the context of Mahalanobis distance learning, the KPCA trick non-trivially has many advantages over the kernel trick; we believe that this consequence is new and is not a consequence of previous works. Also, mathematical tools provided in previous works (Schölkopf & Smola, 2001; Chapelle & Schölkopf, 2001) are not enough to prove the validity of the KPCA trick in this context, and thus the new validation proof of the KPCA trick is needed (see our Theorem 1).

### 3.2 The Kernel-Trick Framework

Given a PSD kernel function  $k(\cdot, \cdot)$  (Schölkopf & Smola, 2001), we denote  $\phi$ ,  $\phi'$  and  $\phi_i$  as mapped data (in a feature space associated with the kernel) of each example  $\mathbf{x}$ ,  $\mathbf{x}'$  and  $\mathbf{x}_i$ , respectively. A (squared) Mahalanobis distance under a matrix  $M$  in the feature space is

$$(\phi_i - \phi_j)^T M (\phi_i - \phi_j) = (\phi_i - \phi_j)^T A^T A (\phi_i - \phi_j). \quad (5)$$

To be consistent with Subsection 2.3, let  $A^T = (\mathbf{a}_1, \dots, \mathbf{a}_d)$ . Denote a (possibly infinite-dimensional) matrix of the mapped training data  $\Phi = (\phi_1, \dots, \phi_n)$ . The main idea of the kernel-trick framework is to parameterize (see representer theorems below)

$$A^T = \Phi U^T, \quad (6)$$

where  $U^T = (\mathbf{u}_1, \dots, \mathbf{u}_d)$ . Substituting  $A$  in Eq. (5) by using Eq. (6), we have

$$(\phi_i - \phi_j)^T M (\phi_i - \phi_j) = (\mathbf{k}_i - \mathbf{k}_j)^T U^T U (\mathbf{k}_i - \mathbf{k}_j),$$

where

$$\mathbf{k}_i = \Phi^T \phi_i = (\langle \phi_1, \phi_i \rangle, \dots, \langle \phi_n, \phi_i \rangle)^T. \quad (7)$$

Now our formula depends only on an inner-product  $\langle \phi_i, \phi_j \rangle$ , and thus the kernel trick can be now applied by using the fact that  $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi_i, \phi_j \rangle$  for a PSD kernel function  $k(\cdot, \cdot)$ . Therefore, the problem of finding the best Mahalanobis distance in the feature space is now reduced to finding the best linear transformation  $U$  of size  $d \times n$ . Nonetheless, it often happens that finding  $U$  is much more troublesome than finding  $A$  in the input space, even their optimization problems look similar, as shown in Section 5.1.

Once we find the matrix  $U$ , the Mahalanobis distance from a new test point  $\mathbf{x}'$  to any input point  $\mathbf{x}_i$  in the feature space can be calculated as follows:

$$\|\phi' - \phi_i\|_M^2 = (\mathbf{k}' - \mathbf{k}_i)^T U^T U (\mathbf{k}' - \mathbf{k}_i), \quad (8)$$

where  $\mathbf{k}' = (k(\mathbf{x}', \mathbf{x}_1), \dots, k(\mathbf{x}', \mathbf{x}_n))^T$ . kNN classification in the feature space can be performed based on Eq. (8).

### 3.3 The KPCA-Trick Framework

As we emphasize above, although the kernel trick framework can be applied to non-linearize the three learners introduced in Section 2, it often happens that finding  $U$  is much more troublesome than finding  $A$  in the input space, even their optimization problems look similar (see Section 5.1). In this section, we develop a *KPCA trick* framework which can be much more conveniently applied to kernelize the three learners.

Denote  $k(\cdot, \cdot)$ ,  $\phi_i$ ,  $\phi$  and  $\phi'$  as in Subsection 3.2. The central idea of the KPCA trick is to represent each  $\phi_i$  and  $\phi'$  in a new “finite”-dimensional space, without any loss of information. Within the framework, a new coordinate of each example is computed “explicitly”, and each example in the new coordinate is then used as the input of any existing Mahalanobis distance learner. As a result, by using the KPCA trick in place of the kernel trick, there is no need to derive new mathematical formulas and no need to implement new algorithms.

To simplify the discussion of KPCA, we assume that  $\{\phi_i\}$  is linearly independent and has its center at the origin, i.e.  $\sum_i \phi_i = 0$  (otherwise,  $\{\phi_i\}$  can be centered by a simple pre-processing step (Shawe-Taylor & Cristianini, 2004, p. 115)). Since we have  $n$  total examples, the span of  $\{\phi_i\}$  has dimensionality  $n$ . Here we claim that each example  $\phi_i$  can be represented as  $\varphi_i \in \mathbb{R}^n$  with respect to a new *orthonormal* basis  $\{\psi_i\}_{i=1}^n$  such that  $\text{span}(\{\psi_i\}_{i=1}^n)$  is the same as  $\text{span}(\{\phi_i\}_{i=1}^n)$  without loss of any information. More precisely, we define

$$\varphi_i = (\langle \phi_i, \psi_1 \rangle, \dots, \langle \phi_i, \psi_n \rangle) = \Psi^T \phi_i. \quad (9)$$

where  $\Psi = (\psi_1, \dots, \psi_n)$ . Note that although we may be unable to numerically represent each  $\psi_i$ , an inner-product of  $\langle \phi_i, \psi_j \rangle$  can be conveniently computed by KPCA (or kernel Gram-Schmidt (Shawe-Taylor & Cristianini, 2004)). Likewise, a new test point  $\phi'$  can be mapped to  $\varphi' = \Psi^T \phi'$ . Consequently, the mapped data  $\{\varphi_i\}$  and  $\varphi'$  are finite-dimensional and can be explicitly computed.

#### 3.3.1 THE KPCA-TRICK ALGORITHM

The KPCA-trick algorithm consisting of three simple steps is shown in Figure 1. NCA, LMNN, DNE and other learners, including those in other settings (e.g. semi-supervised settings), whose kernel versions are previously unknown (Yang et al., 2006; Xing et al., 2003; Chatpatanasiri & Kijsirikul, 2008) can all be kernelized by this simple algorithm. Therefore, it is much more convenient to kernelize a learner by applying the KPCA-trick framework rather than applying the kernel-trick framework. In the algorithm, we denote a Mahalanobis distance learner by `maha` which performs the optimization process shown in Eq. 1 (or Eq. 2) and outputs the best Mahalanobis distance  $M^*$  (or the best linear map  $A^*$ ).

#### 3.3.2 REPRESENTER THEOREMS

Is it valid to represent an infinite-dimensional vector  $\phi$  by a finite-dimensional vector  $\varphi$ ? In the context of SVMs (Chapelle & Schölkopf, 2001), this validity of the KPCA trick is

**Input:** 1. training examples:  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ ,  
 2. new example:  $\mathbf{x}'$ ,  
 3. kernel function:  $k(\cdot, \cdot)$   
 4. Mahalanobis distance learning algorithm: `maha`

**Algorithm:**

- (1) Apply `kPCA`( $k, \{\mathbf{x}_i\}, \mathbf{x}'$ ) such that  $\{\mathbf{x}_i\} \mapsto \{\varphi_i\}$  and  $\mathbf{x}' \mapsto \varphi'$ .
- (2) Apply `maha` with new inputs  $\{(\varphi_i, y_i)\}$  to achieve  $M^*$  or  $A^*$ .
- (3) Perform kNN based on the distance  
 $\|\varphi_i - \varphi'\|_{M^*}$  or  $\|A^* \varphi_i - A^* \varphi'\|$ .

Figure 1: The KPCA-trick algorithm.

easily achieved by straightforwardly extending a proof of an established representer theorem (Schölkopf et al., 2001)<sup>2</sup>. In the context of Mahalanobis distance learning, however, proofs provided in previous works cannot be directly extended. Note that, in the SVM cases considered in previous works, what is learned is a hyperplane, a linear functional outputting a 1-dimensional value. In our case, as shown in Eq. 6, what is learned is a linear map which, in general, outputs a *countably infinite dimensional* vector. Hence, to prove the validity of the KPCA trick in our case, we need some mathematical tools which can handle a countably infinite dimensionality. Below we give our versions of representer theorems which prove the validity of the KPCA trick in the current context.

By our representer theorems, it is the fact that, given an objective function  $f(\cdot)$  (see Eq. (1)), the optimal value of  $f(\cdot)$  based on the input  $\{\phi_i\}$  is equal to the optimal value of  $f(\cdot)$  based on the input  $\{\varphi_i\}$ . Hence, the representation of  $\varphi_i$  can be safely applied. We separate the problem of Mahalanobis distance learning into two different cases. The first theorem covers Mahalanobis distance learners (learning a full-rank linear transformation) while the second theorem covers dimensionality reduction algorithms (learning a low-rank linear transformation).

**Theorem 1.** (*Full-Rank Representer Theorem*) *Let  $\{\tilde{\psi}_i\}_{i=1}^n$  be a set of points in a feature space  $\mathcal{X}$  such that  $\text{span}(\{\tilde{\psi}_i\}_{i=1}^n) = \text{span}(\{\phi_i\}_{i=1}^n)$ , and  $\mathcal{X}$  and  $\mathcal{Y}$  be separable Hilbert spaces. For an objective function  $f$  depending only on  $\{\langle A\phi_i, A\phi_j \rangle\}$ , the optimization*

$$\begin{aligned} \min_A : \quad & f(\langle A\phi_1, A\phi_1 \rangle, \dots, \langle A\phi_i, A\phi_j \rangle, \dots, \langle A\phi_n, A\phi_n \rangle) \\ \text{s.t. } & A : \mathcal{X} \rightarrow \mathcal{Y} \text{ is a bounded linear map ,} \end{aligned}$$

has the same optimal value as,

$$\min_{A' \in \mathbb{R}^{n \times n}} f(\tilde{\varphi}_1^T A'^T A' \tilde{\varphi}_1, \dots, \tilde{\varphi}_i^T A'^T A' \tilde{\varphi}_j, \dots, \tilde{\varphi}_n^T A'^T A' \tilde{\varphi}_n),$$

where  $\tilde{\varphi}_i = (\langle \phi_i, \tilde{\psi}_1 \rangle, \dots, \langle \phi_i, \tilde{\psi}_n \rangle)^T \in \mathbb{R}^n$ .

2. A representer theorem, along with Mercer theorem, is a key ingredient for validating the kernel trick (Schölkopf & Smola, 2001). The origin of the classical representer theorem is dated back to at least 1970s (Kimeldorf & Wahba, 1971).

To our knowledge, mathematical tools provided in the work of Schölkopf and Smola (2001, Chapter 4) are not enough to prove Theorem 1. The proof presented here is a non-straightforward extension of Schölkopf and Smola's work. We also note that Theorem 1, as well as Theorem 2 shown below, is more general than what we discuss above. They justify both the kernel trick (by substituting  $\tilde{\psi}_i = \phi_i$  and hence  $\tilde{\varphi}_i = \mathbf{k}_i$ ) and the KPCA trick (by substituting  $\tilde{\psi}_i = \psi_i$  and hence  $\tilde{\varphi}_i = \varphi_i$ ).

To start proving Theorem 1, the following lemma is useful.

**Lemma 1.** *Let  $\mathcal{X}, \mathcal{Y}$  be two Hilbert spaces and  $\mathcal{Y}$  is separable, i.e.  $\mathcal{Y}$  has a countable orthonormal basis  $\{e_i\}_{i \in \mathbb{N}}$ . Any bounded linear map  $A : \mathcal{X} \rightarrow \mathcal{Y}$  can be uniquely decomposed as  $\sum_{i=1}^{\infty} \langle \cdot, \tau_i \rangle_{\mathcal{X}} e_i$  for some  $\{\tau_i\}_{i \in \mathbb{N}} \subseteq \mathcal{X}$ .*

*Proof.* As  $A$  is bounded, the linear functional  $\phi \mapsto \langle A\phi, e_i \rangle_{\mathcal{Y}}$  is bounded for every  $i$  since, by Cauchy-Schwarz inequality,  $|\langle A\phi, e_i \rangle_{\mathcal{Y}}| \leq \|A\phi\| \|e_i\| \leq \|A\| \|\phi\|$ . By Riesz representation theorem, the map  $\langle A \cdot, e_i \rangle_{\mathcal{Y}}$  can be written as  $\langle \cdot, \tau_i \rangle_{\mathcal{X}}$  for a unique  $\tau_i \in \mathcal{X}$ . Since  $\{e_i\}_{i \in \mathbb{N}}$  is an orthonormal basis of  $\mathcal{Y}$ , for every  $\phi \in \mathcal{X}$ ,  $A\phi = \sum_{i=1}^{\infty} \langle A\phi, e_i \rangle_{\mathcal{Y}} e_i = \sum_{i=1}^{\infty} \langle \phi, \tau_i \rangle_{\mathcal{X}} e_i$ .  $\square$

*Proof.* (Theorem 1) To avoid complicated notations, we omit subscripts such as  $\mathcal{X}, \mathcal{Y}$  of inner products. The proof will consist of two steps. In the first step, we will prove the theorem by assuming that  $\{\tilde{\psi}_i\}_{i=1}^n$  is an orthonormal set. In the second step, we prove the theorem in general cases where  $\{\tilde{\psi}_i\}_{i=1}^n$  is not necessarily orthonormal. The proof of the first step requires an application of Fubini theorem (Lewkeeratiyutkul, 2006).

**Step 1.** Assume that  $\{\tilde{\psi}_i\}_{i=1}^n$  is an orthonormal set. Let  $\{e_i\}_{i=1}^{\infty}$  be an orthonormal basis of  $\mathcal{Y}$ . For any  $\phi' \in \mathcal{X}$ , we have, by Lemma 1,  $A\phi' = \sum_{k=1}^{\infty} \langle \phi', \tau_k \rangle e_k$ . Hence, for each bounded linear map  $A : \mathcal{X} \rightarrow \mathcal{Y}$ , and  $\phi, \phi' \in \text{span}(\{\tilde{\psi}_i\}_{i=1}^n)$ , we have  $\langle A\phi, A\phi' \rangle = \sum_{k=1}^{\infty} \langle \phi, \tau_k \rangle \langle \phi', \tau_k \rangle$ .

Note that Each  $\tau_k$  can be decomposed as  $\tau'_k + \tau_k^{\perp}$  such that  $\tau'_k$  lies in  $\text{span}(\{\tilde{\psi}_i\}_{i=1}^n)$  and  $\tau_k^{\perp}$  is orthogonal to the span. These facts make  $\langle \phi', \tau_k \rangle = \langle \phi', \tau'_k \rangle$  for every  $k$ . Moreover,  $\tau'_k = \sum_{j=1}^n u_{kj} \tilde{\psi}_j$ , for some  $\{u_{k1}, \dots, u_{kn}\} \subset \mathbb{R}^n$ . Hence, we have

$$\begin{aligned}
 \langle A\phi, A\phi' \rangle &= \sum_{k=1}^{\infty} \langle \phi, \tau_k \rangle \langle \phi', \tau_k \rangle = \sum_{k=1}^{\infty} \langle \phi, \tau'_k \rangle \langle \phi', \tau'_k \rangle \\
 &= \sum_{k=1}^{\infty} \langle \phi, \sum_{i=1}^n u_{ki} \tilde{\psi}_i \rangle \langle \phi', \sum_{i=1}^n u_{ki} \tilde{\psi}_i \rangle \\
 &= \sum_{k=1}^{\infty} \sum_{i,j=1}^n u_{ki} u_{kj} \langle \phi, \tilde{\psi}_i \rangle \langle \phi', \tilde{\psi}_j \rangle \\
 (\text{Fubini theorem: explained below}) &= \sum_{i,j=1}^n \left( \sum_{k=1}^{\infty} u_{ki} u_{kj} \right) \langle \phi, \tilde{\psi}_i \rangle \langle \phi', \tilde{\psi}_j \rangle \\
 &= \sum_{i,j=1}^n G_{ij} \langle \phi, \tilde{\psi}_i \rangle \langle \phi', \tilde{\psi}_j \rangle \\
 &= \tilde{\varphi}^T G \tilde{\varphi}' = \tilde{\varphi}^T A'^T A' \tilde{\varphi}'.
 \end{aligned}$$

At the fourth equality, we apply Fubini theorem to swap the two summations. To see that Fubini theorem can be applied at the fourth equality, we first note that  $\sum_{k=1}^{\infty} u_{ki}^2$  is finite

for each  $i \in \{1 \dots n\}$  since

$$\sum_{k=1}^{\infty} u_{ki}^2 = \sum_{k=1}^{\infty} \langle \tilde{\psi}_i, \sum_{j=1}^n u_{kj} \tilde{\psi}_j \rangle \langle \tilde{\psi}_i, \sum_{j=1}^n u_{kj} \tilde{\psi}_j \rangle = \|A\tilde{\psi}_i\|^2 < \infty.$$

Applying the above result together with Cauchy-Schwarz inequality and Fubini theorem for non-negative summation, we have

$$\begin{aligned} \sum_{k=1}^{\infty} \sum_{i,j=1}^n |u_{ki} u_{kj} \langle \phi, \tilde{\psi}_i \rangle \langle \phi', \tilde{\psi}_j \rangle| &= \sum_{i,j=1}^n \sum_{k=1}^{\infty} |u_{ki} u_{kj} \langle \phi, \tilde{\psi}_i \rangle \langle \phi', \tilde{\psi}_j \rangle| \\ &= \sum_{i,j=1}^n |\langle \phi, \tilde{\psi}_i \rangle \langle \phi', \tilde{\psi}_j \rangle| \left( \sum_{k=1}^{\infty} |u_{ki} u_{kj}| \right) \\ &\leq \sum_{i,j=1}^n |\langle \phi, \tilde{\psi}_i \rangle \langle \phi', \tilde{\psi}_j \rangle| \sqrt{\left( \sum_{k=1}^{\infty} u_{ki}^2 \right) \left( \sum_{k=1}^{\infty} u_{kj}^2 \right)} \\ &< \infty. \end{aligned}$$

Hence, the summation converges absolutely and thus Fubini theorem can be applied as claimed above. Again, using the fact that  $\sum_{k=1}^{\infty} u_{ki}^2 < \infty$ , we have that each element of  $G$ ,  $G_{ij} = \sum_{k=1}^{\infty} u_{ki} u_{kj}$ , is finite. Furthermore, the matrix  $G$  is PSD since each of its elements can be regarded as an inner product of two vectors in  $\ell_2$ .

Hence, we finally have that  $\langle A\phi_i, A\phi_j \rangle = \tilde{\varphi}_i^T A'^T A' \tilde{\varphi}_j$ , for each  $1 \leq i, j \leq n$ . Hence, whenever a map  $A$  is given, we can construct  $A'$  such that it results in the same objective function value. By reversing the proof, it is easy to see that the converse is also true. The first step of the proof is finished.

**Step 2.** We now prove the theorem without assuming that  $\{\tilde{\psi}_i\}_{i=1}^n$  is an orthonormal set. Let all notations be the same as in Step 1. Let  $\Psi'$  be the matrix  $(\tilde{\psi}_1, \dots, \tilde{\psi}_n)$ . Define  $\{\psi_i\}_{i=1}^n$  as an orthonormal set such that  $\text{span}(\{\psi_i\}_{i=1}^n) = \text{span}(\{\tilde{\psi}_i\}_{i=1}^n)$  and  $\Psi = (\psi_1, \dots, \psi_n)$  and  $\varphi_i = \Psi^T \phi_i$ . Then, we have that  $\tilde{\psi}_i = \Psi \mathbf{c}_i$  for some  $\mathbf{c}_i \in \mathbb{R}^n$  and  $\Psi' = \Psi C$  where  $C = (\mathbf{c}_1, \dots, \mathbf{c}_n)$ . Moreover, since  $C$  map from an independent set  $\{\psi_i\}$  to another independent set  $\{\tilde{\psi}_i\}$ ,  $C$  is invertible. We then have, for any  $A'$ ,

$$\begin{aligned} \tilde{\varphi}_i^T A'^T A' \tilde{\varphi}_j &= \phi_i^T \Psi' A'^T A' \Psi^T \phi_j \\ &= \phi_i^T \Psi C A'^T A' C^T \Psi^T \phi_j \\ &= \varphi_i^T C A'^T A' C^T \varphi_j \\ &= \varphi_i^T B^T B \varphi_j, \end{aligned}$$

where  $A' C^T = B$  and  $A' = B(C^T)^{-1}$ . Hence, for any  $B$  we have the matrix  $A'$  which gives  $\tilde{\varphi}_i^T A'^T A' \tilde{\varphi}_j = \varphi_i^T B^T B \varphi_j$ . Using the same arguments as in Step 1, we finish the proof of Step 2 and of Theorem 1.  $\square$

**Theorem 2.** (Low-Rank Representer Theorem) Define  $\{\tilde{\psi}_i\}_{i=1}^n$  and  $\tilde{\varphi}_i$  be as in Theorem 1. an objective function  $f$  depending only on  $\{\langle A\phi_i, A\phi_j \rangle\}$ , the optimization

$$\begin{aligned} \min_A : \quad & f(\langle A\phi_1, A\phi_1 \rangle, \dots, \langle A\phi_i, A\phi_j \rangle, \dots, \langle A\phi_n, A\phi_n \rangle) \\ \text{s.t. } & A : \mathcal{X} \rightarrow \mathbb{R}^d \text{ is a bounded linear map ,} \end{aligned}$$

has the same optimal value as,

$$\min_{A' \in \mathbb{R}^{d \times n}} f(\tilde{\varphi}_1^T A'^T A' \tilde{\varphi}_1, \dots, \tilde{\varphi}_i^T A'^T A' \tilde{\varphi}_i, \dots, \tilde{\varphi}_n^T A'^T A' \tilde{\varphi}_n).$$

The proof of Theorem 2 is a generalization of the proofs in previous works (Schölkopf & Smola, 2001, Chap. 4).

*Proof.* (Theorem 2) Let  $\{e_i\}_{i=1}^d$  be the canonical basis of  $\mathbb{R}^d$ , and let  $\phi \in \text{span}\{\tilde{\psi}_1, \dots, \tilde{\psi}_n\}$ . By Lemma 1,  $A\phi = \sum_{i=1}^d \langle \phi, \tau_i \rangle e_i$  for some  $\tau_1, \dots, \tau_d \in \mathcal{X}$ . Each  $\tau_i$  can be decomposed as  $\tau'_i + \tau_i^\perp$  such that  $\tau'_i$  lies in  $\text{span}\{\tilde{\psi}_1, \dots, \tilde{\psi}_n\}$  and  $\tau_i^\perp$  is orthogonal to the span. These facts make  $\langle \phi, \tau_i \rangle = \langle \phi, \tau'_i \rangle$  for every  $i$ . We then have, for some  $u_{ij} \in \mathbb{R}$ ,  $1 \leq i \leq d$ ,  $1 \leq j \leq n$ ,

$$A\phi = \sum_{i=1}^d \langle \phi, \sum_{j=1}^n u_{ij} \tilde{\psi}_j \rangle e_i = \sum_{i=1}^d e_i \sum_{j=1}^n u_{ij} \langle \phi, \tilde{\psi}_j \rangle = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{d1} & \cdots & u_{dn} \end{bmatrix} \begin{bmatrix} \langle \phi, \tilde{\psi}_1 \rangle \\ \vdots \\ \langle \phi, \tilde{\psi}_n \rangle \end{bmatrix} = U\tilde{\varphi}.$$

Since every  $\phi_i$  is in the span, we conclude that  $A\phi_i = U\tilde{\varphi}_i$ . Now, one can easily check that  $\langle A\phi_i, A\phi_j \rangle = \tilde{\varphi}_i^T U^T U \tilde{\varphi}_j$ . Hence, whenever a map  $A$  is given, we can construct  $U$  such that it results in the same objective function value. By reversing the proof, it is easy to see that the converse is also true, and thus the theorem is proven (by renaming  $U$  to  $A'$ ).  $\square$

Note that the proof of Theorem 2 cannot be directly used for proving Theorem 1 (let  $d = \infty$  and  $U \in \mathbb{R}^{\infty \times n}$ , and Theorem 2 is still valid. However, to practically be useful, we need a finite-dimensional linear map. Hence, we must show that  $U^T U \in \mathbb{R}^{n \times n}$  by proving that  $u_{ij} < \infty$  for all  $i, j$ ).

### 3.3.3 REMARKS

1. Note that by Mercer theorem (Schölkopf & Smola, 2001, pp. 37), we can either think of each  $\phi_i \in \ell_2$  or  $\phi_i \in \mathbb{R}^N$  for some positive integer  $N$ , and thus the assumption of Theorem 1 that  $\mathcal{X}$ , as well as  $\mathcal{Y}$ , is separable Hilbert space is then valid. Also, both theorems require that the objective function of a learning algorithm must depend only on  $\{\langle A\phi_i, A\phi_j \rangle\}_{i,j=1}^n$  or equivalently  $\{\langle \phi_i, M\phi_j \rangle\}_{i,j=1}^n$ . This condition is, actually, not a strict condition since learners in literatures have their objective functions in this form (Chen et al., 2005; Goldberger et al., 2005; Globerson & Roweis, 2006; Weinberger et al., 2006; Yang et al., 2006; Sugiyama, 2006; Yan et al., 2007; Zhang et al., 2007; Torresani & Lee, 2007).
2. Note that the two theorems stated in this section do not require  $\{\tilde{\psi}_i\}$  to be an orthonormal set. However, there is an advantage of the KPCA trick which restricts  $\tilde{\psi}_i = \psi_i$  as in Eq. (9); this will be discussed in Sect. 5.1.
3. A running time of each learner strongly depends on the dimensionality of the input data. As recommended by Weinberger et al. (2006), it can be helpful to first apply a dimensionality reduction algorithm such as PCA before performing a learning process: the learning process can be tremendously speed up by retaining only, says, the 200 largest-variance principal components of the input data. In the KPCA trick framework illustrated

in Figure 1, dimensionality reduction can be performed without any extra work as KPCA is already applied at the first place.

4. The stronger version of Theorem 2 can be achieved by inserting a regularizer into the objective function of a (kernelized) Mahalanobis distance learner as stated in Theorem 3. For compact notations, we use the fact that  $A$  is representable by  $\{\tau_i\}$  as shown in Lemma 1.

**Theorem 3.** (*Strong Representer Theorem*) Define  $\{\tilde{\psi}_i\}_{i=1}^n$  and  $f$  be as in Theorem 1. For monotonically increasing functions  $g_i$ , let

$$h(\tau_1, \dots, \tau_d, \phi_1, \dots, \phi_n) = f(\langle \tau_1, \phi_1 \rangle, \dots, \langle \tau_i, \phi_j \rangle, \dots, \langle \tau_n, \phi_d \rangle) + \sum_{i=1}^d g_i(\|\tau_i\|).$$

Any optimal set of linear functionals

$$\begin{aligned} & \arg \min_{\{\tau_i\}} h(\tau_1, \dots, \tau_d, \phi_1, \dots, \phi_n) \\ & \text{s.t. } \forall i \tau_i : \mathcal{X} \rightarrow \mathbb{R} \text{ is a bounded linear functional} \end{aligned}$$

must admit the representation of  $\tau_i = \sum_{j=1}^n u_{ij} \tilde{\psi}_j \quad (i = 1, \dots, d)$ .

The proof of this result is very similar to that of (Schölkopf & Smola, 2001, Theorem 4.2) so that we omit its details here. In fact, to prove the validation of KDNE, we need this strong representer theorem (see the case of KPCA in Schölkopf and Smola (2001, pp.92)).

To apply Theorem 3 to our framework, we can simply view  $A = (\tau_1, \dots, \tau_d)^T$ . If each  $g_i$  is the square function, then regularizer becomes  $\sum_{i=1}^d \|\tau_i\|^2 = \|A\|_{HS}$  where  $\|\cdot\|_{HS}$  is the Hilbert-Schmidt (HS) norm of an operator. If each  $\tau_i$  is finite-dimensional, the HS norm is reduced to the Frobenius norm  $\|\cdot\|_F$ . Here, we allow the HS norm of a bounded linear operator to take a value of  $\infty$ . For the kernel trick (by substituting  $\tilde{\psi}_i = \phi_i$ ), the result above states that any optimal  $\{\tau_i\}$  must be represented by  $\{\Phi \mathbf{u}_i\}$ . Therefore, using the same notation as Subsection 3.2, we have

$$\sum_{i=1}^d g_i(\|\tau_i\|) = \sum_{i=1}^d \|\tau_i\|^2 = \sum_{i=1}^d \mathbf{u}_i^T \Phi^T \Phi \mathbf{u}_i = \sum_{i=1}^d \mathbf{u}_i^T K \mathbf{u}_i = \text{trace}(U K U^T).$$

This regularizer is first appeared in the work of Globerson and Roweis (2006). Similarly, for the KPCA trick (by substituting  $\tilde{\psi}_i = \psi_i$ ), any optimal  $\{\tau_i\}$  must be represented by  $\{\Psi \mathbf{u}_i\}$  and, using the fact that  $\Psi^T \Psi = I$ , we have  $\sum_{i=1}^d \|\tau_i\|^2 = \text{trace}(U U^T) = \|U\|_F^2$ .

By adding the regularizer,  $\text{trace}(U K U^T)$  or  $\|U\|_F^2$ , into existing objective functions, we have a new class of learners, namely, *regularized Mahalanobis distance learners* such as regularized KNCA (RKNCA), regularized KLMNN (RKLMNN) and regularized KDNE (RKDNE). Our framework can be further extended into a problem in semi-supervised settings by adding more complicated functions of  $g_i(\cdot)$  such as *manifold regularizers*, see e.g. Chatpatanasiri and Kijsirikul (2008). We plan to investigate effects of using various types of regularizers in the near future.

## 4. Selection of a Kernel Function

The problem of selecting an efficient kernel function is central to all kernel machines. All previous works on Mahalanobis distance learners use exhaustive methods such as cross validation to select a kernel function. In this section, we investigate a possibility to automatically construct a kernel which is appropriate for a Mahalanobis distance learner. In the first part of this section, we consider a popular method called *kernel alignment* (Lanckriet et al., 2004; Zhu et al., 2005) which is able to learn, from a training set, a kernel in the form of  $k(\cdot, \cdot) = \sum_i \alpha_i k_i(\cdot, \cdot)$  where  $k_1(\cdot, \cdot), \dots, k_m(\cdot, \cdot)$  are pre-chosen base kernels. In the second part of this section, we investigate a simple method which constructs an unweighted combination of base kernels,  $\sum_i k_i(\cdot, \cdot)$  (henceforth referred to as an *unweighted kernel*). A theoretical result is provided to support this simple approach. Kernel constructions based on our two approaches require much shorter running time when comparing to the standard cross validation approach.

### 4.1 Kernel Alignment

Here, our kernel alignment formulation belongs to the class of quadratic programs (QPs) which can be solved more efficiently than the formulations proposed by Lanckriet et al. (2004) and Zhu et al. (2005) which belong to the class of semidefinite programs (SDPs) and quadratically constrained quadratic programs (QCQPs), respectively (Boyd & Vandenberghe, 2004).

To use kernel alignment in classification problems, the following assumption is central: for each couple of examples  $\mathbf{x}_i, \mathbf{x}_j$ , the ideal kernel  $k(\mathbf{x}_i, \mathbf{x}_j)$  is  $Y_{ij}$  (Guermeur et al., 2004) where

$$Y_{ij} = \begin{cases} +1, & \text{if } y_i = y_j, \\ \frac{-1}{p-1}, & \text{otherwise,} \end{cases}$$

and  $p$  is the number of classes in the training data. Denoting  $Y$  as the matrix having elements of  $Y_{ij}$ , we then define the *alignment* between the kernel matrix  $K$  and the ideal kernel matrix  $Y$  as follows:

$$\text{align}(K, Y) = \frac{\langle K, Y \rangle_F}{\|K\|_F \|Y\|_F}, \quad (10)$$

where  $\langle \cdot, \cdot \rangle_F$  denotes the Frobenius inner-product such that  $\langle K, Y \rangle_F = \text{trace}(K^T Y)$  and  $\|\cdot\|_F$  is the Frobenius norm induced by the Frobenius inner-product.

Assume that we have  $m$  kernel functions,  $k_1(\cdot, \cdot), \dots, k_m(\cdot, \cdot)$  and  $K_1, \dots, K_m$  are their corresponding Gram matrices with respect to the training data. In this paper, the kernel function obtained from the alignment method is parameterized in the form of  $k(\cdot, \cdot) = \sum_i \alpha_i k_i(\cdot, \cdot)$  where  $\alpha_i \geq 0$ . Note that the obtained kernel function is guaranteed to be positive semidefinite. In order to learn the best coefficients  $\alpha_1, \dots, \alpha_m$ , we solve the following optimization problem:

$$\{\alpha_1, \dots, \alpha_m\} = \arg \max_{\alpha_i \geq 0} \text{align}(K, Y), \quad (11)$$

where  $K = \sum_i \alpha_i K_i$ . Note that as  $K$  and  $Y$  are PSD,  $\langle K, Y \rangle_F \geq 0$ . Since both the numerator and denominator terms in the alignment equation can be arbitrary large, we can

simply fix the numerator to 1. We then reformulate the problem as follows:

$$\begin{aligned}
 \arg \max_{\alpha_i \geq 0, \langle K, Y \rangle_F = 1} \text{align}(K, Y) &= \arg \min_{\alpha_i \geq 0, \langle K, Y \rangle_F = 1} \|K\|_F \|Y\|_F \\
 &= \arg \min_{\alpha_i \geq 0, \langle K, Y \rangle_F = 1} \|K\|_F^2 \\
 &= \arg \min_{\alpha_i \geq 0, \sum_i \alpha_i \langle K_i, Y \rangle_F = 1} \sum_{i,j} \alpha_i \alpha_j \langle K_i, K_j \rangle_F.
 \end{aligned}$$

Defining a PSD matrix  $S$  whose elements  $S_{ij} = \langle K_i, K_j \rangle_F$ , a vector  $\mathbf{b} = (\langle K_1, Y \rangle_F, \dots, \langle K_m, Y \rangle_F)^T$  and a vector  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_m)^T$ , we then reformulate Eq. (11) as follows:

$$\boldsymbol{\alpha} = \arg \min_{\alpha_i \geq 0, \boldsymbol{\alpha}^T \mathbf{b} = 1} \boldsymbol{\alpha}^T S \boldsymbol{\alpha}. \quad (12)$$

This optimization problem is a QP and can be efficiently solved (Boyd & Vandenberghe, 2004); hence, we are able to learn the best kernel function  $k(\cdot, \cdot) = \sum_i \alpha_i k_i(\cdot, \cdot)$  efficiently.

Since the magnitudes of the optimal  $\alpha_i$  are varied due to  $\|K_i\|_F$ , it is convenient to use  $k'_i(\cdot, \cdot) = k_i(\cdot, \cdot) / \|K_i\|_F$  and hence  $K'_i = K_i / \|K_i\|_F$  in the derivation of Eq. (12). We define  $S'$  and  $\mathbf{b}'$  similar to  $S$  and  $\mathbf{b}$  except that they are based on  $K'_i$  instead of  $K_i$ . Let

$$\boldsymbol{\gamma} = \arg \min_{\gamma_i \geq 0, \boldsymbol{\gamma}^T \mathbf{b}' = 1} \boldsymbol{\gamma}^T S' \boldsymbol{\gamma}. \quad (13)$$

It is easy to see that the final kernel function  $k(\cdot, \cdot) = \sum_i \gamma_i k'_i(\cdot, \cdot)$  achieved from Eq. (13) is not changed from the kernel achieved from Eq. (12).

Note that we can further modify Eq. (12) to enforce sparseness of  $\boldsymbol{\alpha}$  and improve a speed of an algorithm by minimizing an upper bound of  $\|K\|_F$  instead of minimizing the exact quantity so that the optimization formula belongs to the class of linear programs (LPs) instead of QPs.

$$\min_{\alpha_i \geq 0, \langle K, Y \rangle_F = 1} \|K\|_F \leq \min_{\alpha_i \geq 0, \langle K, Y \rangle_F = 1} \|\text{vec}(K)\|_1 \quad (14)$$

where  $\text{vec}(\cdot)$  denotes a standard ‘‘vec’’ operator converting a matrix to a vector (Minka, 1997). By using a standard trick for an absolute-valued objective function (Boyd & Vandenberghe, 2004), Eq. (14) can be solved by linear programming. Note that the above optimization algorithm of minimizing the upper bound of a desired objective function is similar to the popular support vector machines where the hinge loss is minimized instead of the 0/1 loss.

## 4.2 Unweighted Kernels

In this subsection, we show that a very simple kernel  $k'(\cdot, \cdot) = \sum_i k_i(\cdot, \cdot)$  is theoretically efficient, no less than a kernel obtained from the alignment method. Denote  $\phi_i^k$  as a mapped vector of an original example  $\mathbf{x}_i$  by a map associated with a kernel  $k(\cdot, \cdot)$ . The main idea of the contents presented in this section is the following simple but useful result.

**Proposition 1.** *Let  $\{\alpha_i\}$  be a set of positive coefficients,  $\alpha_i > 0$  for each  $i$ , and let  $k_1(\cdot, \cdot), \dots, k_m(\cdot, \cdot)$  be base PSD kernels and  $k(\cdot, \cdot) = \sum_i \alpha_i k_i(\cdot, \cdot)$  and  $k'(\cdot, \cdot) = \sum_i k_i(\cdot, \cdot)$ . Then, there exists an invertible linear map  $B$  such that  $B : \phi_i^{k'} \rightarrow \phi_i^k$  for each  $i$ .*

*Proof.* Without loss of generality, we will concern here only the case of  $m = 2$ ; the cases such that  $m > 2$  can be proven by induction. Let  $\mathcal{H}_i \oplus \mathcal{H}_j$  be a direct sum of  $\mathcal{H}_i$  and  $\mathcal{H}_j$  where its inner product is defined by  $\langle \cdot, \cdot \rangle_{\mathcal{H}_i} + \langle \cdot, \cdot \rangle_{\mathcal{H}_j}$  and let  $\{\phi_i^{(j)}\} \subset \mathcal{H}_j$  denote a mapped training set associated with the  $j^{th}$  base kernel. Then we can view  $\phi_i^k = (\sqrt{\alpha_1}\phi_i^{(1)}, \sqrt{\alpha_2}\phi_i^{(2)}) \in \mathcal{H}_i \oplus \mathcal{H}_j$  since

$$\begin{aligned}\langle \phi_i^k, \phi_j^k \rangle &= k(\mathbf{x}_i, \mathbf{x}_j) = \alpha_1 k_1(\mathbf{x}_i, \mathbf{x}_j) + \alpha_2 k_2(\mathbf{x}_i, \mathbf{x}_j) \\ &= \langle \sqrt{\alpha_1}\phi_i^{(1)}, \sqrt{\alpha_1}\phi_j^{(1)} \rangle + \langle \sqrt{\alpha_2}\phi_i^{(2)}, \sqrt{\alpha_2}\phi_j^{(2)} \rangle \\ &= \langle (\sqrt{\alpha_1}\phi_i^{(1)}, \sqrt{\alpha_2}\phi_i^{(2)}), (\sqrt{\alpha_1}\phi_j^{(1)}, \sqrt{\alpha_2}\phi_j^{(2)}) \rangle.\end{aligned}$$

Similarly, we can also view  $\phi_i^{k'} = (\phi_i^{(1)}, \phi_i^{(2)}) \in \mathcal{H}_i \oplus \mathcal{H}_j$ . Let  $I_j$  be the identity map in  $\mathcal{H}_j$ . Then,

$$B = \begin{bmatrix} \sqrt{\alpha_1}I_1 & 0 \\ 0 & \sqrt{\alpha_2}I_2 \end{bmatrix}.$$

Since  $\infty > \alpha_1, \alpha_2 > 0$  and  $B$  is bounded (the operator norm of  $B$  is  $\max(\sqrt{\alpha_1}, \sqrt{\alpha_2})$ ),  $B$  is invertible.  $\square$

Now suppose we apply the kernel  $k(\cdot, \cdot) = \sum_i \alpha_i k_i(\cdot, \cdot)$  obtained from the kernel alignment method to a Mahalanobis distance learner and an optimal transformation  $A^*$  is returned. Let  $f(\cdot)$  be an objective function which depends only on an inner product  $\langle A\phi_i, A\phi_j \rangle$  (as assumed in Theorems 1 and 2). Since, from Proposition 1,  $\langle A^*\phi_i^k, A^*\phi_j^k \rangle = \langle A^*B\phi_i^{k'}, A^*B\phi_j^{k'} \rangle$ , we have

$$f^* \equiv f \left( \left\{ \langle A^*\phi_i^k, A^*\phi_j^k \rangle \right\} \right) = f \left( \left\{ \langle A^*B\phi_i^{k'}, A^*B\phi_j^{k'} \rangle \right\} \right).$$

Thus, by applying a training set  $\{\phi_i^{k'}\}$  to a learner who tries to minimize  $f(\cdot)$ , a learner will return a linear map with the objective value less than or equal to  $f^*$  (because the learner can at least return  $A^*B$ ). Notice that because  $B$  is invertible, the value  $f^*$  is in fact optimal. Consequently, the following claim can be stated: “there is no need to apply the methods which learn  $\{\alpha_i\}$ , e.g. the kernel alignment method, *at least in theory*, because learning with a simple kernel  $k'(\cdot, \cdot)$  also results in a linear map having the same optimal objective value”. However, *in practice*, there can be some differences between using the two kernels  $k(\cdot, \cdot)$  and  $k'(\cdot, \cdot)$  due to the following reasons.

- **Existence of a local solution.** As some optimization problems are not convex, there is no guarantee that a solver is able to discover a global solution within a reasonable time. Usually, a learner discovers only a local solution, and hence two learners based on  $k(\cdot, \cdot)$  and  $k'(\cdot, \cdot)$  will not give the same solution. KNCA belongs to this case.

- **Non-existence of the unique global solution.** In some optimization problems, there can be many different linear maps having the same optimal values  $f^*$ , and hence there is no guarantee that two learners based on  $k(\cdot, \cdot)$  and  $k'(\cdot, \cdot)$  will give the same solution. KLMNN is an example of this case.

- **Size constraints.** Because of a size constraint such as  $AA^T = I$  used in KDNE, our arguments used in the previous subsection cannot be applied, i.e., given that  $A^{*T}A^* = I$ ,

there is no guaranteed that  $(A^*B)(A^*B)^T = I$ . Hence,  $A^*B$  may not be an optimal solution of a learner based on  $k'(\cdot, \cdot)$ .

- **Preprocessing of target neighbors.** The behavior of some learners depends on their preprocesses. For example, before learning takes place, the KLMNN and KDNE algorithms have to specify the target neighbors of each point (by specifying a value of  $w_{ij}$ ). In a case of using the KPCA trick, this specification is based on the Euclidean distance with respect to a selected kernel (see Subsection 5.1.2 and Proposition 3). In this case, the Euclidean distance with respect to an aligned kernel  $k(\cdot, \cdot)$  (which already used some information of a training set) is more appropriate than the Euclidean distance with respect to an unweighted kernel  $k'(\cdot, \cdot)$ .

- **Zero coefficients.** In the above proposition we assume  $\alpha_i > 0$  for all  $i$ . Often, the alignment algorithm return  $\alpha_i = 0$  for some  $i$ . Define  $A^*$  and  $f^*$  as above. Following the same line of the proof of Proposition 1, in the cases that the alignment method gives  $\alpha_i = 0$  for some  $i$ , it can be easily shown that a learner with a kernel  $k'(\cdot, \cdot)$  will return a linear map with its objective value better than or equal to  $f^*$ .

Since constructing  $k'(\cdot, \cdot)$  is extremely easy,  $k'(\cdot, \cdot)$  is a very attractive choice to be used in kernelized algorithms.

## 5. Demonstrations

In this section, the advantages of the KPCA trick over the kernel trick are demonstrated. After that, we conduct extensive experiments to illustrate the performance of kernelized algorithms, especially for those applying the kernel construction methods described in the previous section.

### 5.1 KPCA Trick versus Kernel Trick

To understand the advantages of the KPCA trick over the kernel trick, it is best to derive a kernel trick formula for each algorithm and see what have to be done in order to implement a kernelized algorithm applying the kernel trick. In this section, we define  $\{\phi_i\}$  and  $\Phi$  as in Section 3.2.

#### 5.1.1 KNCA

As noted in Sect. 2.1, in order to minimize the objective of NCA and KNCA, we need to derive gradient formulas, and the formula of  $\partial f^{KNCA} / \partial A$  is (Goldberger et al., 2005):

$$-2A \sum_i \left( p_i \sum_k p_{ik} \phi_{ik} \phi_{ik}^T - \sum_{j \in c_i} p_{ij} \phi_{ij} \phi_{ij}^T \right) \quad (15)$$

where for brevity we denote  $\phi_{ij} = \phi_i - \phi_j$ . Nevertheless, since  $\phi_i$  may lie in an infinite dimensional space, the above formula cannot be always implemented in practice. In order to implement the kernel-trick version of KNCA, users need to prove the following proposition which is not stated in the original work of Goldberger et al. (2005):

**Proposition 2.**  $\partial f^{KNCA} / \partial A$  can be formulated as  $V\Phi^T$  where  $V$  depends on  $\{\phi_i\}$  only in the form of  $\langle \phi_i, \phi_j \rangle = k(\mathbf{x}_i, \mathbf{x}_j)$ , and thus we can compute all elements of  $V$ .

*Proof.* Define a matrix  $B_i^\phi = (0, 0, \dots, \phi, \dots, 0, 0)$  as a matrix with its  $i^{th}$  column is  $\phi$  and zero vectors otherwise. Denote  $\mathbf{k}_{ij} = \mathbf{k}_i - \mathbf{k}_j$ . Substitute  $A = U\Phi^T$  to Eq. (15) we have

$$\begin{aligned} \frac{\partial f^{KNCA}}{\partial A} &= -2U \sum_i \left( p_i \sum_k p_{ik} \mathbf{k}_{ik} \phi_{ik}^T - \sum_{j \in c_i} p_{ij} \mathbf{k}_{ij} \phi_{ij}^T \right) \\ &= -2U \sum_i \left( p_i \sum_k p_{ik} (B_i^{\mathbf{k}_{ik}} - B_k^{\mathbf{k}_{ik}}) - \sum_{j \in c_i} p_{ij} (B_i^{\mathbf{k}_{ij}} - B_j^{\mathbf{k}_{ij}}) \right) \Phi^T \\ &= V\Phi^T, \end{aligned}$$

which completes the proof.  $\square$

Therefore, at the  $i^{th}$  iteration of an optimization step of a gradient optimizer, we needs to update the current best linear map as follows:

$$\begin{aligned} A^{(i)} &= A^{(i-1)} + \epsilon \frac{\partial f^{KNCA}}{\partial A} = (U^{(i-1)} + \epsilon V^{(i-1)})\Phi^T \\ &= U^{(i)}\Phi^T, \end{aligned} \quad (16)$$

where  $\epsilon$  is a step size. The kernel-trick formulas of KNCA are thus finally achieved. However, we emphasize that the process of proving Proposition 2 and Eq. (16) is not trivial and may be tedious and difficult for non-experts as well as practitioners who focus their tasks on applications rather than theories. Moreover, since the formula of  $\partial f^{KNCA}/\partial A$  is significantly different from  $\partial f^{NCA}/\partial A$ , users are required to re-implement KNCA (even they already possess an NCA implementation) which is again not at all convenient. In contrast, we note that all these difficulties are disappeared if the KPCA trick algorithm consisting of three simple steps shown in Fig. 1 is applied instead of the kernel trick.

There is another advantage of using the KPCA trick on KNCA<sup>3</sup>. By the nature of a gradient optimizer, it takes a large amount of time for NCA and KNCA to converge to a local solution, and thus a method of speeding up the algorithms is needed. As recommended by Weinberger et al. (2006), it can be helpful to first apply PCA before performing a learning process: the learning process can be tremendously speed up by retaining only, says, the 100 largest-variance principal components of the input data. In the KPCA trick framework, no extra work is required for this speed-up task as KPCA is already applied at the first place.

### 5.1.2 KLMNN

Similar to KNCA, the online-available code of LMNN<sup>4</sup> employs a gradient based optimization, and thus new gradient formulas in the feature space has to be derived and new implementation has to be done in order to apply the kernel trick. On the other hand, by applying the KPCA trick, the original LMNN code can be immediately used.

There is another advantage of the KPCA trick on LMNN: LMNN requires a specification of  $w_{ij}$  which is usually based on the quantity  $\|\mathbf{x}_i - \mathbf{x}_j\|$ . Thus, it makes sense that  $w_{ij}$  should

3. We slightly modify the code of Charless Fowlkes: <http://www.cs.berkeley.edu/~fowlkes/software/nca/>

4. <http://www.weinbergerweb.net/Downloads/LMNN.html>

be based on  $\|\phi_i - \phi_j\| = \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) - 2k(\mathbf{x}_i, \mathbf{x}_j)}$  with respect to the feature space of KLMNN, and hence, with the kernel trick, users have to modify the original code in order to appropriately specify  $w_{ij}$ . In contrast, by applying the KPCA trick which restricts  $\{\psi_i\}$  to be an orthonormal set as in Eq. (9), we have the following proposition:

**Proposition 3.** *Let  $\{\psi_i\}_{i=1}^n$  be an orthonormal set such that  $\text{span}(\{\psi_i\}_{i=1}^n) = \text{span}(\{\phi_i\}_{i=1}^n)$  and  $\varphi_i = (\langle \phi_i, \psi_1 \rangle, \dots, \langle \phi_i, \psi_n \rangle)^T \in \mathbb{R}^n$ , then  $\|\varphi_i - \varphi_j\|^2 = \|\phi_i - \phi_j\|^2$  for each  $1 \leq i, j \leq n$ .*

*Proof.* Since we work on a separable Hilbert space  $\mathcal{X}$ , we can extend the orthonormal set  $\{\psi_i\}_{i=1}^n$  to  $\{\psi_i\}_{i=1}^\infty$  such that  $\text{span}(\{\psi_i\}_{i=1}^\infty)$  is  $\mathcal{X}$  and  $\langle \phi_i, \psi_j \rangle = 0$  for each  $i = 1, \dots, n$  and  $j > n$ . Then, by an application of the Parseval identity (Lewkeeratiyutkul, 2006),

$$\begin{aligned} \|\phi_i - \phi_j\|^2 &= \sum_{k=1}^{\infty} \langle \phi_i - \phi_j, \psi_k \rangle^2 = \sum_{k=1}^n \langle \phi_i - \phi_j, \psi_k \rangle^2 \\ &= \|\varphi_i - \varphi_j\|^2. \end{aligned}$$

The last equality comes from Eq.(9).  $\square$

Therefore, with the KPCA trick, the target neighbors  $w_{ij}$  of each point is computed based on  $\|\varphi_i - \varphi_j\| = \|\phi_i - \phi_j\|$  without any modification of the original code.

### 5.1.3 KDNE

By applying  $A = U\Phi^T$  and defining the gram matrix  $K = \Phi^T\Phi$ , we have the following proposition.

**Proposition 4.** *The kernel-trick formula of KDNE is the following minimization problem:*

$$U^* = \arg \min_{UKU^T=I} \text{trace}(UK(D - W)KU^T). \quad (17)$$

Note that this kernel-trick formula of KDNE involves a *generalized eigenvalue problem* instead of a plain eigenvalue problem involved in DNE. As a consequence, we face a *singularity* problem, i.e. if  $K$  is not full-rank, the constraint  $UKU^T = I$  cannot be satisfied. Using elementary linear algebra, it can be shown that  $K$  is not full-rank if and only if  $\{\phi_i\}$  is not linearly independent, and this condition is not highly improbable. Sugiyama (2006), Yu and Yang (2001), and Yang and Yang (2003) suggest methods to cope with the singularity problem in the context of Fisher discriminant analysis which may be applicable to KDNE. Sugiyama (2006) recommends to use the constraint  $U(K + \epsilon I)U^T = I$  instead of the original constraint; however, an appropriate value of  $\epsilon$  has to be tuned by cross validation which is time-consuming. Alternatively, Yu and Yang (2001) and Yang and Yang (2003) propose more complicated methods of directly minimizing an objective function in the null space of the constraint matrix so that the singularity problem is explicitly avoided.

We note that a KPCA-trick implementation of KDNE does not have this singularity problem as only a plain eigenvalue problem has to be solved. Moreover, as in KLMNN, applying the KPCA trick instead of the kernel trick to KDNE avoid the tedious task of modifying the original code to appropriately specify  $w_{ij}$  in the feature space.

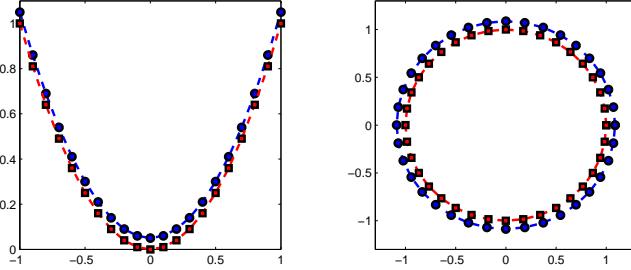


Figure 2: Two synthetic examples where NCA, LMNN and DNE cannot learn any efficient Mahalanobis distances for kNN. Note that in each example, data in each class lie on a simple non-linear 1-dimensional subspace (which, however, cannot be discovered by the three learners). In contrast, the kernel versions of the three algorithms (using the 2<sup>nd</sup>-order polynomial kernel) can learn very efficient distances, i.e., the non-linear subspaces are discovered by the kernelized algorithms.

Table 1: The average accuracy with standard deviation of NCA and their kernel versions. On the bottom row, the win/draw/lose statistics of each kernelized algorithm comparing to its original version is drawn.

NAME	NCA	KNCA	ALIGNED KNCA	UNWEIGHTED KNCA
BALANCE	$0.89 \pm 0.03$	<b><math>0.92 \pm 0.01</math></b>	<b><math>0.92 \pm 0.01</math></b>	<b><math>0.91 \pm 0.03</math></b>
BREAST CANCER	$0.95 \pm 0.01$	<b><math>0.97 \pm 0.01</math></b>	<b><math>0.96 \pm 0.01</math></b>	<b><math>0.96 \pm 0.02</math></b>
GLASS	$0.61 \pm 0.05$	<b><math>0.69 \pm 0.02</math></b>	<b><math>0.69 \pm 0.04</math></b>	<b><math>0.68 \pm 0.04</math></b>
IONOSPHERE	$0.83 \pm 0.04$	<b><math>0.94 \pm 0.03</math></b>	<b><math>0.92 \pm 0.02</math></b>	<b><math>0.90 \pm 0.03</math></b>
IRIS	$0.96 \pm 0.03$	$0.96 \pm 0.01$	$0.95 \pm 0.03$	$0.96 \pm 0.02$
MUSK2	$0.87 \pm 0.02$	<b><math>0.90 \pm 0.01</math></b>	<b><math>0.88 \pm 0.02</math></b>	$0.87 \pm 0.02$
PIMA	$0.68 \pm 0.02$	<b><math>0.71 \pm 0.02</math></b>	$0.67 \pm 0.03$	<b><math>0.69 \pm 0.01</math></b>
SATELLITE	$0.82 \pm 0.02$	<b><math>0.84 \pm 0.01</math></b>	<b><math>0.84 \pm 0.01</math></b>	$0.82 \pm 0.02$
YEAST	$0.47 \pm 0.02$	<b><math>0.50 \pm 0.01</math></b>	<b><math>0.49 \pm 0.02</math></b>	$0.47 \pm 0.02$
WIN/DRAW/LOSE	-	8/1/0	7/0/2	5/4/0

## 5.2 Numerical Experiments

On page 8 of the LMNN paper (Weinberger et al., 2006), Weinberger et al. gave a comment about KLMNN: ‘*as LMNN already yields highly nonlinear decision boundaries in the original input space, however, it is not obvious that “kernelizing” the algorithm will lead to significant further improvement*’. Here, before giving experimental results, we explain why “kernelizing” the algorithm can lead to significant improvements. The main intuition behind the kernelization of “Mahalanobis distance learners for the kNN classification algorithm” lies in the fact that non-linear boundaries produced by kNN (with or without Mahalanobis distance) is usually helpful for problems with multi-modalities; however, the non-linear boundaries of kNN is sometimes not helpful when data of the same class stay on a low-dimensional non-linear manifold as shown in Figure 2.

In this section, we conduct experiments on NCA, LMNN, DNE and their kernel versions on nine real-world datasets to show that (1) it is really the case that the kernelized algorithms usually outperform their original versions on real-world datasets, and (2) the performance of linearly combined kernels achieved by the two methods presented in Section 4 are comparable to kernels which are exhaustively selected, but the kernel alignment method requires much shorter running time.

Table 2: The average accuracy with standard deviation of LMNN and their kernel versions.

NAME	LMNN	KLMNN	ALIGNED KLMNN	UNWEIGHTED KLMNN
BALANCE	0.84 $\pm$ 0.04	<b>0.87 <math>\pm</math> 0.01</b>	<b>0.88 <math>\pm</math> 0.02</b>	<b>0.85 <math>\pm</math> 0.01</b>
BREAST CANCER	0.95 $\pm$ 0.01	<b>0.97 <math>\pm</math> 0.01</b>	<b>0.97 <math>\pm</math> 0.00</b>	<b>0.97 <math>\pm</math> 0.00</b>
GLASS	0.63 $\pm$ 0.05	<b>0.69 <math>\pm</math> 0.04</b>	<b>0.69 <math>\pm</math> 0.04</b>	<b>0.66 <math>\pm</math> 0.05</b>
IONOSPHERE	0.88 $\pm$ 0.02	<b>0.95 <math>\pm</math> 0.02</b>	<b>0.94 <math>\pm</math> 0.02</b>	<b>0.94 <math>\pm</math> 0.02</b>
IRIS	0.95 $\pm$ 0.02	<b>0.96 <math>\pm</math> 0.02</b>	0.95 $\pm$ 0.02	<b>0.97 <math>\pm</math> 0.01</b>
MUSK2	0.80 $\pm$ 0.03	<b>0.93 <math>\pm</math> 0.01</b>	<b>0.88 <math>\pm</math> 0.02</b>	<b>0.86 <math>\pm</math> 0.02</b>
PIMA	0.68 $\pm$ 0.02	<b>0.71 <math>\pm</math> 0.02</b>	<b>0.72 <math>\pm</math> 0.02</b>	0.67 $\pm$ 0.03
SATELLITE	0.81 $\pm$ 0.01	<b>0.85 <math>\pm</math> 0.01</b>	<b>0.84 <math>\pm</math> 0.01</b>	<b>0.83 <math>\pm</math> 0.02</b>
YEAST	0.47 $\pm$ 0.02	<b>0.48 <math>\pm</math> 0.02</b>	<b>0.54 <math>\pm</math> 0.02</b>	<b>0.50 <math>\pm</math> 0.02</b>
WIN/DRAW/LOSE	-	9/0/0	8/1/0	8/0/1

Table 3: The average accuracy with standard deviation of DNE and their kernel versions.

NAME	DNE	KDNE	ALIGNED KDNE	UNWEIGHTED KDNE
BALANCE	0.79 $\pm$ 0.02	<b>0.90 <math>\pm</math> 0.01</b>	<b>0.83 <math>\pm</math> 0.02</b>	<b>0.85 <math>\pm</math> 0.03</b>
BREAST CANCER	0.96 $\pm$ 0.01	<b>0.97 <math>\pm</math> 0.01</b>	0.96 $\pm$ 0.01	0.96 $\pm$ 0.02
GLASS	0.65 $\pm$ 0.04	<b>0.70 <math>\pm</math> 0.03</b>	<b>0.69 <math>\pm</math> 0.04</b>	0.65 $\pm$ 0.03
IONOSPHERE	0.87 $\pm$ 0.02	<b>0.95 <math>\pm</math> 0.02</b>	<b>0.95 <math>\pm</math> 0.02</b>	<b>0.93 <math>\pm</math> 0.03</b>
IRIS	0.95 $\pm$ 0.02	<b>0.97 <math>\pm</math> 0.02</b>	<b>0.96 <math>\pm</math> 0.02</b>	<b>0.96 <math>\pm</math> 0.03</b>
MUSK2	0.89 $\pm$ 0.02	<b>0.91 <math>\pm</math> 0.01</b>	0.89 $\pm$ 0.02	0.84 $\pm$ 0.03
PIMA	0.67 $\pm$ 0.02	<b>0.69 <math>\pm</math> 0.02</b>	<b>0.70 <math>\pm</math> 0.03</b>	<b>0.70 <math>\pm</math> 0.02</b>
SATELLITE	0.84 $\pm$ 0.01	<b>0.85 <math>\pm</math> 0.01</b>	<b>0.85 <math>\pm</math> 0.01</b>	0.81 $\pm$ 0.02
YEAST	0.40 $\pm$ 0.05	<b>0.48 <math>\pm</math> 0.01</b>	<b>0.47 <math>\pm</math> 0.04</b>	<b>0.52 <math>\pm</math> 0.02</b>
WIN/DRAW/LOSE	-	9/0/0	7/2/0	5/2/2

To measure the generalization performance of each algorithm, we use the nine real-world datasets obtained from the UCI repository (Asuncion & Newman, 2007): BALANCE, BREAST CANCER, GLASS, IONOSPHERE, IRIS, MUSK2, PIMA, SATELLITE and YEAST. Following previous works, we randomly divide each dataset into training and testing sets. By repeating the process 40 times, we have 40 training and testing sets for each dataset. The generalization performance of each algorithm is then measured by the average test accuracy over the 40 testing sets of each dataset. The number of training data is 200 except for GLASS and IRIS where we use 100 examples because these two datasets contain only 214 and 150 total examples, respectively.

Following previous works, we use the 1NN classifier in all experiments. In order to kernelize the algorithms, three approaches are applied to select appropriate kernels:

- Cross validation (KNCA, KLMNN and KDNE).
- Kernel alignment (ALIGNED KNCA, ALIGNED KLMNN and ALIGNED KDNE).
- Unweighted combination of base kernels (UNWEIGHTED KNCA, UNWEIGHTED KLMNN and UNWEIGHTED KDNE).

For all three methods, we consider scaled RBF base kernels (Schölkopf & Smola, 2001, p. 216),  $k(x, y) = \exp(-\frac{\|x-y\|^2}{2D\sigma^2})$  where  $D$  is the dimensionality of input data. Twenty one based kernels specified by the following values of  $\sigma$  are considered: 0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1, 2.5, 5, 7.5, 10, 25, 50, 75, 100, 250, 500, 750, 1000. all kernelized algorithms are implemented by the KPCA trick illustrated in Figure 1. As noted

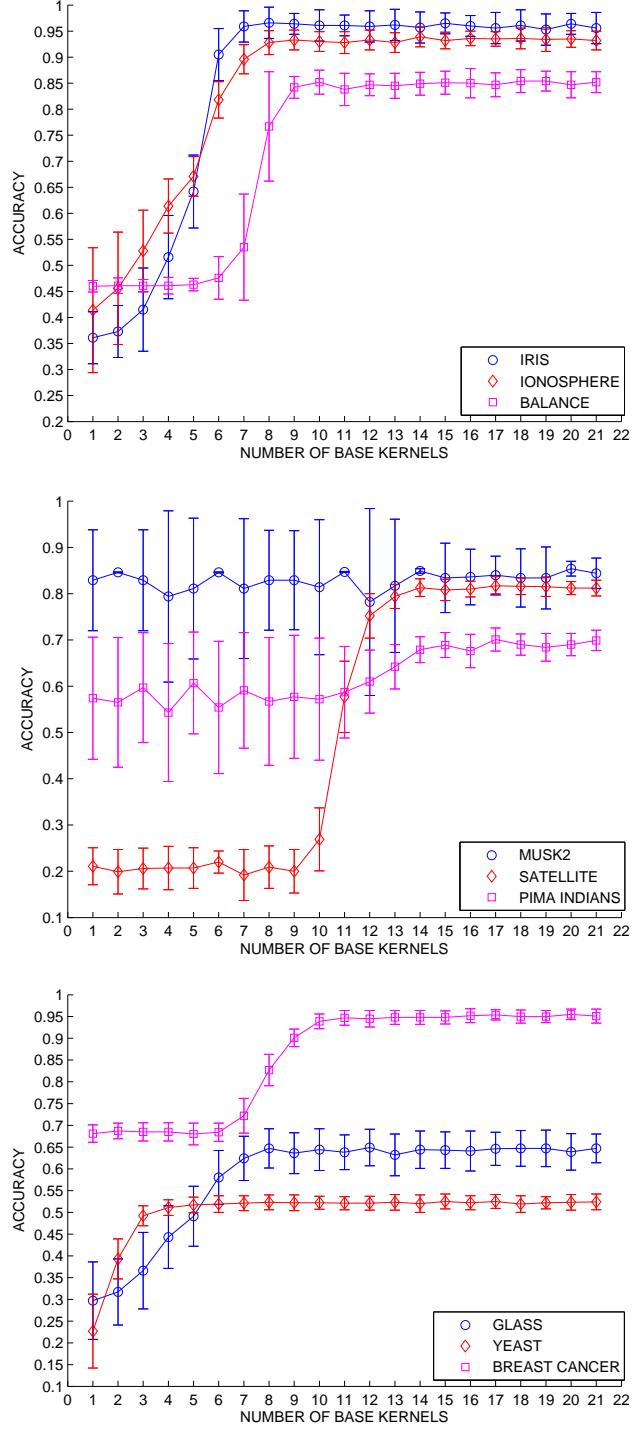


Figure 3: This figure illustrates performance of UNWEIGHTED KDNE with different number of base kernels. It can be observed from the figure that the generalization performance of UNWEIGHTED KDNE will be eventually stable as we add more and more base kernels.

in Subsection 4.2, the main problem of using the unweighted kernel to algorithms such as KLMNN and KDNE is that the Euclidean distance with respect to the unweighted kernel

is not informative and thus should not be used to specify target neighbors of each point. Therefore, in cases of KLMNN and KDNE which apply the unweighted kernel, we employ the Euclidean distance with respect to the input space to specify target neighbors. We slightly modify the original codes of LMNN and DNE to fulfill this desired specification.

The experimental results are shown in Tables 1, 2 and 3. From the results, it is clear that the kernelized algorithms usually improve the performance of their original algorithms. The kernelized algorithms applying cross validation obtain the best performance. They outperform the original methods in 26 out of 27 datasets. The other two kernel versions of the three original algorithms also have satisfiable performance. The kernelized algorithms applying kernel alignment outperform the original algorithms in 22 datasets and obtain an equal performance in 3 datasets. Only 2 out of 27 datasets where the original algorithms outperform the kernel algorithms applying kernel alignment. Similarly, the kernelized algorithms applying the unweighted kernel outperform the original algorithms in 18 datasets and obtain an equal performance in 6 datasets. Only 3 out of 27 datasets where the original algorithms outperform the kernel algorithms applying the unweighted kernel.

We note that although the cross validation method usually gives the best performance, the other two kernel construction methods provide comparable results in much shorter running time. For each dataset, a run-time overhead of the kernelized algorithms applying cross validation is of several hours (on Pentium IV 1.5GHz, Ram 1 GB) while run-time overheads of the kernelized algorithms applying aligned kernels and the unweighted kernel are about minutes and seconds, respectively, for each dataset. Therefore, in time-limited circumstance, it is attractive to apply an aligned kernel or an unweighted kernel.

Note that the kernel alignment method are not appropriate for a multi-modal dataset in which there may be several clusters of data points for each class since, from eq. (10), the function  $\text{align}(K, Y)$  will attain the maximum value if and only if all points of the same class are collapsed into a single point. This may be one reason which explains why cross validated kernels give better results than results of aligned kernels in our experiments. Developing a new kernel alignment algorithm which suitable for multi-modality is currently an open problem.

Comparing generalization performance induced by aligned kernels and the unweighted kernel, algorithms applying aligned kernels perform slightly better than algorithms applying the unweighted kernel. With little overhead and satisfiable performance, however, the unweighted kernel is still attractive for algorithms, like NCA (in contrast to LMNN and DNE), which are not required a specification of target neighbors  $w_{ij}$ . Since Euclidean distance with respect to the unweighted kernel is usually not appropriate for specifying  $w_{ij}$ , an KPCA-trick application of algorithms like LMNN and DNE may still require some re-programming.

As noted in the previous section, aligned kernels usually does not use all base kernels ( $\alpha_i = 0$  for some  $i$ ); in contrast, the unweighted kernel uses all base kernels ( $\alpha_i = 1$  for all  $i$ ). Hence, as described in Section 4.2, the feature space corresponding to the unweighted kernel usually contains the feature space corresponding to aligned kernels. Therefore, we may informally say that the feature space induced by the unweighted kernel is “larger” than ones induced by aligned kernels.

Since a feature space which is too large can lead to overfitting, one may wonder whether or not using the unweighted kernel leads to overfitting. Figure 3 shows that overfitting

indeed does not occur. For compactness, we show only the results of UNWEIGHTED KDNE. In the experiments shown in this figure, base kernels are adding in the following order: 0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1, 2.5, 5, 7.5, 10, 25, 50, 75, 100, 250, 500, 750, 1000. It can be observed from the figure that the generalization performance of UNWEIGHTED KDNE will be eventually stable as we add more and more base kernels. Also, It can be observed that 10 - 14 base kernels are enough to obtain stable performance. It is interesting to further investigate an overfitting behavior of a learner by applying methods such as a bias-variance analysis (James, 2003) and investigate whether it is appropriate or not to apply an “adaptive resampling and combining” method (Breiman, 1998) to improve the classification performance of a supervised mahalanobis distance learner.

## 6. Summary

We have presented general frameworks to kernerlize Mahalanobis distance learners. Three recent algorithms are kernelized as examples. Although we have focused only on the supervised settings, the frameworks are clearly applicable to learners in other settings as well, e.g. a semi-supervised learner. Two representer theorems which justify both our framework and those in previous works are formally proven. The theorems can also be applied to Mahalanobis distance learners in unsupervised and semi-supervised settings. Moreover, we present two methods which can be efficiently used for constructing a good kernel function from training data. Although we have concentrated only on Mahalanobis distance learners, our kernel construction methods can be indeed applied to all kernel classifiers. Numerical results over various real-world datasets showed consistent improvements of kernelized learners over their original versions.

### ACKNOWLEDGEMENTS

This work is supported by Thailand Research Fund. We thank Wicharn Lewkeeratiyutkul who taught us the theory of Hilbert space. We also thank Prasertsak Pungprasertying for preparing some datasets in the experiments.

## References

Asuncion, A., & Newman, D. J. (2007). UCI machine learning repository..

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics*, 26, 801–823.

Chapelle, O., & Schölkopf, B. (2001). Incorporating Invariances in Nonlinear SVMs. *NIPS*.

Chatpatanasiri, R., & Kijsirikul, B. (2008). Spectral Methods for Linear and Non-Linear Semi-Supervised Dimensionality Reduction. *Arxiv preprint arXiv:0804.0924*.

Chen, H.-T., Chang, H.-W., & Liu, T.-L. (2005). Local discriminant embedding and its variants. In *CVPR*, Vol. 2.

Globerson, A., & Roweis, S. (2006). Metric learning by collapsing classes. *NIPS*.

Goldberger, J., Roweis, S., Hinton, G., & Salakhutdinov, R. (2005). Neighbourhood components analysis. *NIPS*.

Guermeur, Y., Lifchitz, A., & Vert, R. (2004). A kernel for protein secondary structure prediction. *Kernel Methods in Computational Biology*.

James, G. M. (2003). Variance and bias for general loss functions. *Machine Learning*, 51, 115–135.

Kimeldorf, G., & Wahba, G. (1971). Some Results on Tchebycheffian Spline Functions. *Journal of Mathematical Analysis and Applications*, 33, 82–95.

Lanckriet, G. R. G., Cristianini, N., Bartlett, P., Ghaoui, L. E., & Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *JMLR*, 5, 27–72.

Lewkeeratiyutkul, W. (2006). *Lecture Notes on Real Analysis I-II*. Available online at <http://pioneer.netserv.chula.ac.th/~wicharn/2301622/>.

Minka, T. (1997). Old and new matrix algebra useful for statistics. See [www.stat.cmu.edu/minka/papers/matrix.html](http://www.stat.cmu.edu/minka/papers/matrix.html).

Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. In *COLT*, pp. 416–426. Springer-Verlag.

Schölkopf, B., & Smola, A. J. (2001). *Learning with Kernels*.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Sugiyama, M. (2006). Local fisher discriminant analysis for supervised dimensionality reduction. In *ICML*.

Torresani, L., & Lee, K. (2007). Large margin components analysis. *NIPS*.

von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 395–416.

Weinberger, K., Blitzer, J., & Saul, L. (2006). Distance metric learning for large margin nearest neighbor classification. *NIPS*.

Xing, E. P., Ng, A. Y., Jordan, M. I., & Russell, S. (2003). Distance metric learning with application to clustering with side-information. *NIPS*.

Yan, S., Xu, D., Zhang, B., Zhang, H.-J., Yang, Q., & Lin, S. (2007). Graph embedding and extensions: A general framework for dimensionality reduction. *PAMI*, 29(1).

Yang, J., & Yang, J. Y. (2003). Why can LDA be performed in PCA transformed space?. *Pattern Recognition*, 36, 563–566.

Yang, L., Jin, R., Sukthankar, R., & Liu, Y. (2006). An efficient algorithm for local distance metric learning. In *AAAI*.

Yu, H., & Yang, J. (2001). A direct LDA algorithm for high-dimensional data - with application to face recognition. *Pattern Recognition*, 34, 2067–2070.

Zhang, W., Xue, X., Sun, Z., Guo, Y.-F., & Lu, H. (2007). Optimal dimensionality of metric space for classification. In *ICML*.

Zhu, X., Kandola, J., Ghahramani, Z., & Lafferty, J. (2005). Nonparametric transforms of graph kernels for semi-supervised learning. *NIPS*.